C语言用户参考手册

C函数库指南

第一章　引言

§1.1　概貌

C语言是一种通用的程序设计语言，它的效率、经济性以及可移植性是众所周知的。这些优点使得几乎任何一种程序设计都可以把C视为一种选择，而且大量事实还证明了C在系统程序设计方面特别有用，因为程序员可以用C写出速度快、结构紧凑的程序，并且可以把这些程序移植到其他系统上。在许多情况下，好的C程序可以在速度上与用汇编语言书写的程序相媲美，而且具有易于维护和可读性好的优点。

尽管C具有效率高、功能强的特点，相对来说，它毕竟还是一种较小的语言。C不包含执行诸如输入、输出、存贮分配、屏幕操作以及过程控制等任务的内部函数，程序员必须根据运行库来执行这些任务。这种设计增加了C的适应性和紧凑性，因为语言是相对地受到某些限制的，所以它不使用一个具体的程序设计模型。运行时的例行程序提供所需的功能，这就允许程序员在需要的时候编一些用于特殊目的的子程序。

这种设计还把语言的特性和具体C语言实现的某些环境分离开来，于是，这对企望写出移植性好的编码的程序员来说是一个帮助。语言的严格定义使得它不依赖于任何操作系统或机器，同时，程序员也可以简单地添加依赖于系统的例程以充分地利用特定的机器。

C语言一些重要的性质如下：

- 为从逻辑上和有效地控制程序流程以及鼓励结构程序设计方法的使用，C
  了循环、条件控制结构的全集。
- C提供了一组数量极多的运行符。许多C运行对应于常用的机器指令，这
  翻译成机器代码提供了方便。各种各样的运算符使得程序员能够以最少的编
  清楚地说明不同类型的运算。
- C数据类型包括几种长度的整数，以及单精度和双精度的浮点数。程序员
  定义更为复杂的数据类型，如数组和结构，以适应程序的需要
- C程序员可以说明指向变量和函数的指针。指向一个实体
  的机器地址，巧妙地使用指针可以极大地提高程序的效
  物理机器的方式访问实体。C还支持指针的算术运符，即
  操内存地址。
- C的预处理器，即正文处理器，在编译之前施作用

来说，这一特点的最有用处的是，程序常数的定义、用更快的类似宏的函数调用的替换以及条件编译。预处理器不只限于 C 文件，它也可以用于任何正文文件。

C 是一种灵活的语言，它把许多东西都留给程序员。为了这一目的，C 在许多方面，例如类型转换，使用了很少的限制条件。这常常是一种很好的机制，但是，为理解程序将怎样动作，程序员必须完全熟悉语言的定义。

1.2 关于这本手册

《XENIX C 语言参考手册》定义了 Microsoft 公司所实现的 C 语言，它的目的是为已经具备 C 或其它程序设计语言经验的程序员提供一本参考手册。在此，我们假定程序员已经具备了程序设计的基础知识。

适用于 MicrosoftC 程序的运行时的库函数在《XENIX C 库参手册》中予以讨论。至于在用户的系统上如何编译和连接 C 程序，也请参阅《XENIX C 库参考手册》，这本手册还提供了在用户的系统上关于 C 实现的某些信息。

《XENIX C 语言参考手册》按如下方式组织：

第一章，"引言"，介绍了手册的内容安排以及所使用的约定

第二章，"C 的基本元素"，描述了用于 C 程序中的字母、数字和符号，以及于编译器有特殊意义的字符的组合。

第三章，"程序结构"，描述了 C 程序的组成成分和结构以及如何组织 C 源文件。

第四章，"说明"，描述了如何指定 C 的变量、函数和用户定义的类型的属性。C 提供了一些预先定义好了的数据类型，并且允许程序员说明集成类型和指针。

第五章，"表达式和赋值"，描述了构成 C 的表达式及赋值的运算符对象和运算符，还讨论了在表达式求值时所进行的类型转换和出现的付作用。

第六章，"语句"，描述了 C 的语句，语言用于控制程序的执行。

第七章，"函数"，讨论了 C 函数的特点，特别地，本章说了如何定义、说明和调用函数以及如何描述函数的参数和返回值。

第八章，"预处理指示"，描述了由 C 的预处理器所识别的命令。C 的预处理器是在编译之前自动装入的一个正文处理器。

附录 A，"区别"，列举了 Microsoft 的 C 和《The C progamming Language》一书的附录 A 描述的 C 语言之间的区别，该书由 Brian W. Kernighan 和 Dennis M. Ritchig 合著，由 tice-Hall 公司于 1978 年出版。

附录 B，"语法概要"，列出了 Microsoft C 语言的语法。

本书的其余部分说明贯穿于全书所使用的一些约定。

命令使用了一些约定：

字必须如所示那样输入的命令，选择项，标志或程序名。黑体字，全程变量，标准类、常数、关键字或 C 库所使用的标识了解一个给定的回函数，参阅《XENIX 参考手册》的"按字

典排序的清单"使可以找到手册中描述该函数的位置。)

italics 斜体字表示文件名,这与库的包含文件名(即 stdio.h)和其他文件名(即/etc/ttys)相称。

当在正文中提及时,斜体字表示提供变量和函数的特殊的标识符。

斜体字表示用户命令的子程序(用户命令的子程序后面跟以开括号和闭括号())。斜体字还表示在正文中要强调的东西。

CAPITAL 大写表示环境变量的合字(如 TZ 和 PATH)。

SMALL CAPITAL 小一号的大写表示键的名字或键的序列(如 RETURN)。

[] 方括号表示其中的项是可以选择的,若用户不使用该选择项,程序将按其省缺值行动。

… 省略号表示前面的项可以重复任何次。

"" 引号表示技术术语的第一次使用。引号表示对一个字而不是命令的引用。

〈文字〉 功能基本上同于斜体字。

第二章　C的基本元素

§2.1 简介

本章描述C语言的基本元素，这些元素是构成C程序的名字，数字和字符。本章特别对下列元素加以阐明：

- 字符集
- 常数
- 标识符
- 关键字
- 注解
- 语法单位

字符集

于C程序中的两个字符集，C字符集和可表示字符集。C字符集包括字有特殊含义的标点，C程序就是由C字符集中的字符组合成有意义

一个子集。可表示字符集包括所有字母、数字以及用户可

以以图形方式用一个字符表示出来的符号。字符集的范围取决于所使用的终端、监视器或字符设备的类型。

除了串、字符常数和注解可以使用任何可表示字符以外，C 程序只能使用 C 字符集中的字符。C 字符集中的每个字符对 C 编译器来说都有其确切的含义，当编译器遇到对字符的错误使用或使用了不属于 C 字符集中的字符时，它将产生错误信息。

下面各小节描述 C 字符集中的字符和符号，并说明如何使用以及什么时候使用之。

§2.2.1 字母和数字

C 的字符集包括英语的大小写字母和十个阿拉伯数字：

大写英语字母：

A B C D E F G H I J K L M O P Q R

S T U V W X Y Z

小写英语字母：

a b c d e f g h i j k l m n o p q r

s t u v w x y z

十进制数字：

0 1 2 3 4 5 6 7 8 9

这些字母和数字可以用来组成在本章后面的部分中所描写的常数、标识符和关键字。

C 编译器区分大小写字母。如果用小写的“a”代表一给定的项，那么就不能用大写的“A”来替代之，而必须用小写。

§2.2.2 空白字符

空格、制表符、换行、回车符符号都称为空白字符，因为输出时它们在字与字之间、行与行之间都起到类似于空格的功能。这些字符把用户定义的项，如常数和标识符，从程序的其他项中区分开来。

除非把空白字符用在字符常数或串常数中，C 编译器是忽略这些字符的，这就是说程序员可以使用额外的空白符增加程序的可读性。注解(见 2.6 节)也视被为空白字符。

§2.2.3 标点和特殊字符

C 字符集中的标点和特殊字符用于许多目的，从组织程序的正文到定义到要求编译器执行的任务或指定被编译的程序要完成的任务。表 2.1 列出了这些字符：

表 2.1 标点和特殊字符

字符	名字	字符	名字
,	逗号	!	惊叹号
.	点	|	竖线
;	分号	/	斜线

:	问号	\	反斜线
?	问号	~	否定号
'	单引号	=	下划线
"	双引号	#	数字号
(	左括号	%	百分号
)	右括号	&	和号
[	左方括号	^	脱字号
]	右方括号	*	星
{	左花括号	-	减号
}	右花括号	=	等于号
<	小于号	+	加号
>	大于号		

对于C编译器来说，这些字符都有其特殊的含意。在本手册中我们描述了它们的用途。可表示字符集中那些未出现在该表中的标点字符仅能用在串常数、字符常数和注解中。

§2.2.4 转义序列

转义序列是表示串和字符常数中空白字符和非图形字符的特殊的字符组合。典型的用途是用它们来指明动作，例如回车或制表，或者是提供有特殊含义的特殊符号的文字表示，例如双引号(")字符。转义序列由反斜线后面跟以一个字母或数字的排列组成。表2.2列出了C语音的转义序列：

表2.2 转义序列

转义序列	名字
\n	换行
\t	水平制表

\v	垂直制表
\b	回退一格
\r	回车
\f	走行
\'	单引号
\"	双引号
\\	反斜线
\ddd	ASCII 字符的八进制表示
\xddd	ASCII 字符的十六进制表示

如果反斜线出现在未出现在上表中的字符之前，那么反斜线就被忽略过去，而该字符就是它的字面表示。例如，在串或字符中，"\c"就是表示"c"

序列"\ddd"和"\xdd"允许用三位八进制数字或两位十六进制数字给出 ASCII 字符集中的任何字符。例如回车符可以用"\0"和"\x0"给出。

在八进制转义序列中只能出现八进制数字，并且至少要有一位数字，但可以少于三位数字，例如，回退一格字符可以由"\10"给出。类似地，十六进制序列应至少包括一位数字，但第二个数字可以省去，回退一格字符的十六进制转义字符可以写为"\x8"。但是，在串中使用八进制或十六进制转义序列时完整地给出转义序列则更为安全一些，否则，在如果序列后紧跟着的字符是八进制数字或十六进制数字时，该字符就可验被解释成序列的一部分。

转义序列使得用户把控制字符送到显示设备上，例如"\033"常用作终端打印机控制命令的第一字符。

非图形字符必须用转义序表示，这种字符出现在 C 程序中时将产生难以预料的结果。

用于引入转义序列的反斜线还可以在串或预处理器定义中当作续行符使用。当换行符跟在反斜线之后时它将被忽略，而下一行被视为上一行的继续。

§2.2.5 运算符

运算符是指明如何把值进行传送和赋值的特殊字符的组合，编译器这种字符的组合视为一体，称为 token，即语法单位(见 2.7 节)。

表 2.3 列出了构成 C 运算符的字符并给出了每个运算符的名称。用户必须准确地按表中所列的那样使用运算符，多字符的运算符之间不能插入空白字符。运算符 sizeof 不包括在表中，它是关键字而不是符号。

表 2.3 运算符

运算符	名字

!	逻辑 NOT
~	逐位求补
+	加
−	减
*	乘
/	除
%	求余数
≪	向左移位
≫	向右移位
<	小于
<=	小于等于
>	大于
>=	大于等于
==	等于
!=	不等于
&	逐位求 AND，求地址
|	逐位求相容 OR
^	逐位求排斥 OR
&&	逻辑 AND
||	逻辑 OR
,	顺序求值
?:	条件式
++	递增
−−	递减
=	简单赋值
+=	加赋值
−=	减赋值
*=	赋值
/=	除赋值

%=	求余数赋值
>>=	右移位赋值
<<=	左移位赋值
&=	逐位求 AND 赋值
|=	逐位求相容 OR 赋值
^=	逐位求排斥 OR 赋值

[注]@条件运算是三元运算符，而不是一个多字符的运算。条件表达式的形式为：表达式？表达式：表达式

关于每个运算符更完整的说明，参见第五章，“表达式和赋值”。

§2.3 常数

常数是在程序中可以作为值使用的数字、字符或字符串，常数的值不因执行的环境不同而改变。

C 语言有四种常数：整数常数，浮点数常数，字符常数。以下各部分说明各类常数的格式

§2.3.1 整数常数

整数常数是表示一个整数值的十进制、八进制或十六进制数。十进制常数的形式为：

〈digits〉

其中〈digits〉是一个或多个十进制数字(从 0 到 9)。

八进制常数的形式为：

0x〈odigits〉

其中〈odigits〉是一个多个用进制数字(从 0 到 7)，前导零是必须的。

十六进制常数的形式为：

0〈hdigits〉

其中〈hdigits〉是一个或多个十进制数字(从 0 到 9. 以及从 a 到 f 和从 A 到 F)，前导零是必须的，而且必须跟以“x”。

整数常数中，数字间不能有空白字符出现。表 2.4 列举了整数常数形式：

表 2.4 整数常数

十进制常数	八进制常数	十六进制常数
10	012	0_xa 或 0_xA
132	0204	0_x84
32179	07663	0_x7dB3 或 0_x7DB3

整数常数总是表示正值。若需在负值，则可以把负号(－)放在常数之前形成一表示负

值的表达式，而负号被视为一个算术运算符。

每个整数常数根据它的值都有一个类型，常数的类型决定了它出现在表达式中或用以形成负常数时将进行的转换类型，十进制常数被认为是带符号的数值，并给以类型 int 或者 long。

八进制和十六年制常数也中为 int 或者 long 类型的，但它们不象其他有符号数那样，在类型转换中不进行符号扩充。

程序员可以告诉编译器让一个常数具有 long 类型，方法意在常数后面附加上"l"或"L"。表 2.5 列举了类型为 long 整数常数

表 2.5 长整数常数

十进制常数	八进制常数	十六进制常数
10L	012L	0xaL 或 0xAL
79l	0115l	0xfl 或 0x4FL

类型的说明在第四章，"说明"中给出，类型转换规则在第五章，"表达式和赋值"中给出。

§ 2.3.2 浮点数常数

浮点数常数是表示一个有符号的实数的十进制数值，有符号的实数有整数、小数和指数三部分。浮点数常数的形式为：

[digits][. digits][E[－]digits]

其中 digits 是一个或多个十进制数字(从 0 到 9)，E(或 e)是指数的符号。小数点以前的数字(值的整数部分)或小数点以后的数字(值的小数部分)均可以省去，但二者不能同时省去。指数由指数符号后面跟以有可能为负值的整数值构成。仅当给出指数时才能省去小数点。不能用空白字符把常数中数字或字符隔开。

浮点数常数总是表示正值，当需要负值时，可以把负号(－)置于常数之前形成一个具有负值的浮点常数表达式，而负号被视为算术运算符。

在下例中，我们列举了浮点常数和表达式的一些形式：

15.75

1.575E1

1575e－2

－0.0025

－2.5e－3

25E－4

浮点数常数的整数的部分可以省去，举例如下：

.75

.0075e2

－.125

－.175E－2

所有浮点数常数都具类型 double。

§2.3.3 字符常数

字符常数是括在单引号之内的字母、数字、标点符号或转义序列，字符常数的值就是它本身。字符常数中不能出现多于一个的字符或转义序列。

字符常数的形式为：

'〈字符〉'

其中〈字符〉可以是可表示字符集中(包括任何转义序列)除了单引号(')或反斜线(\)和换行符以外的任何字符。为了把单引号或反斜线用作字符常数，应该按表 2.6 所示那样在它前面再加一个反斜线，而用转义序列'\n'来表示换行字符。

表 2.6 字符常数举例

常数	值
'a'	小写的字母
'?'	问号
'\b'	回退一格字符
'\x1B'	ASCII 的 ESC 字符
'\''	单引号
'\\'	反斜线

字符常数的类型为 char，在类型转换时进行符号扩充(见 5.7 节)。

§2.3.4 串常数

串常数是括在双引号之内的字母、数字和符号的序列。串常数为字符数组，数组的每一个元素是一个字符的值。

串常数的形式为：

"〈字符组〉"

其中〈字符组〉是可表示字符集中除双引号(")、反斜线(\)和换行符以外的一个或多个字符。为在串中使用换行符，可以在换行之前加一个反斜线，反斜线使得换行符被略。这就允许一个串常数可以点据多于一行的位置。例如，

串：

```
"Long strings can be bro\
ken into two pieces."
```

等同于串：

```
"Long strings can be broken into two piceces."
```

为在串中使用双引号或反斜线，可以在它们之前再加一个反斜线。如下例所示：

```
"This is a string literal."
"Enter a number between 1 and 100\n Or press Return."
"First\\Second."
```

"\"Yes,I do \"she said. "

注意,串常数中可以有转义序列(如\n 和\")。

串中的字符依次存贮在内存中一块连续的区域内,并且把空字符(\0)自动地附加到串的尾部。程序中的每个串都被视为不同的项,如果程序中有两个相同的串,那么它们各自都有自已的存贮区域。

串常数的类型为 char[],即串是元素类型为 char 的数组,数组中元素的个数是串中的字符的个数加 1,因为最后一个字符之后的空字符也被计算在内。

§2.4 标识符

标识符是用户为程序中所使用变量、函数和标号所起的名字。用户可以创建一个标识符,方法是在说明部分把它和变量或函数联系在一起,用户也可以在以后使用该标识符以指代给定的项(说明在第四章,"说明"中介绍)。

标识符是以字母或下划线(-)开头的字母、数字或划线序列,标识符中可以有任意数目的字符,但只有前 31 个对编译器来说才有意义。(其他需要编译器的输出作为输入的程序,如连接程序,可能使用更少的字符)。使用前导的下划线必须小心,因为这时有可能和隐含的系统子程序发生名字冲突,从而产生错误。

下面是一些标识符的例子:

```
j
cnt
templ
top.of.page
skip2
```

C 编译器区分大小写英文字母,因此,可以说明拼写方法一样但大小写不完全一样的不同的标识符。例如,下列标识符是互不相同的:

```
add
ADD
Add
aDD
```

C 编译器不允许和关键字拼写方法和大小写都完全一样的标识符。关键字在 2.5 节给出。

连接程序可能要进一步限制全程可见符号中的字符的类型和数目,而且不象编译器那样,连接程序还可能不区分大小写。关于连接程序所使用的名字规则请查阅相应的连接程序的文档资料。

§2.5 关键字

关键字是对 C 编译器来说有特殊含意的预先定义的标识符,程序项的名字不能与下面所列出的关键字发生冲突:

auto	default	float	register	swith
berak	do	fkr	return	tupedef

case	double	goto	short	union
char	else	if	sizof	unsigned
const	enum	int	struct	woid
continue	exten	struct	while	

不能重定义关键字，但是可以在编译之前用预处理器指示说明用以替换关键字的文字（见第八章，“预处理器指示”）。

关键字 const 留作将来使用，至今尚未在语言中实现。

下列标识符有可能在具体的实现中视为关键字（详细情况请查阅系统说明）：

far

fortran

huge

near

pascal

§2.6 注解

注解是被编译器视为空白字符并予以略去的字符序列，它的形式为：

/*〈字符组〉*/

这里〈字符组〉可以是可表示字符集中的任何字，包括换行符的任意组合，但不能包含“*/，”这就是说注解可以占据一行以上的位置，但是不能嵌套。

注解典型的用处是说明源程序中的语句或动作，它可以出现在空白字符所能出现的任何地方。既然编译器对注解的字符不予理睬，关键字就可以出现在注解中而不会产生任何错误。

下面是一些注解的例子：

```
/* Comments can seperate and document
lines of a progam. */
/* Comments can contain keywords such as for
and while. */
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
Comment can occupy several lines.
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
```

因为注解不能嵌套，所以下例是错误的：

```
/* You canot,/* nest */ /comments */
```

编译器把“nest”之后的第一个“*/”识别成注解的结束，而处理其余的正文就会产生错误。

为了压缩程序的大模块或包含注解的程序段的编译，可以使用预处理器指示 #if，而不使用注解（见 8.4 节）。